

Kubernetes

Lev Epshtein



Lev Epshtein

Solution Architect

lev@opsguru.io

- Docker overview
- Kubernetes
 - Core Concepts
 - Pods
 - Services
 - Labels
 - Namespaces, Annotations
 - Deployments
 - DaemonSets
 - ReplicaSets
 - Jobs
 - ConfigMaps and Secrets
 - Stateful set





kubernetes



What is Kubernetes?

“Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.”

Kubernetes

Kubernetes has a number of features. It can be thought of as:

- a container platform
- a microservices platform
- a portable cloud platform and a lot more.
- Kubernetes provides a container-centric management environment. It orchestrates computing, networking, and storage infrastructure on behalf of user workloads. This provides much of the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), and enables portability across infrastructure providers.
- K8S Allows developers / system operators to cut to the cord and truly run a container-centric dev / microservice environment



Kubernetes

Core Concepts & Components



Kubernetes Concepts

Understanding K8S system and abstraction

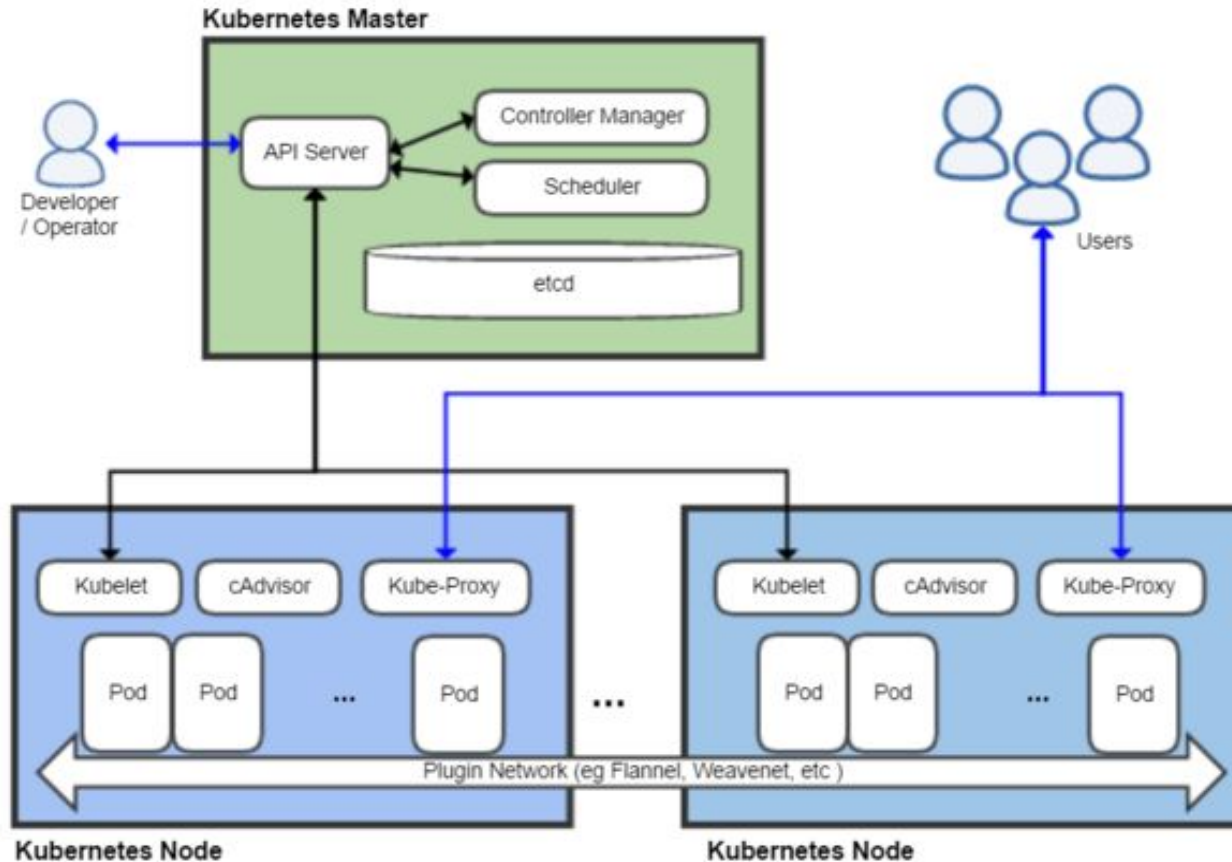
To work with Kubernetes, we use Kubernetes API objects to describe our cluster's desired state:

- what applications or other workloads we want to run
- What container images they use, the number of replicas, what network and disk resources we want to make available.
- Setting our desired state by creating objects using the Kubernetes API (typically via the command-line interface - kubectl)

Once we've set our desired state,

Kubernetes Control Plane works to make the cluster's current state match the desired state

Kubernetes



K8S: Master Components

Master components provide the cluster's control plane

- **Kube-apiserver** - Validates and configures data for the api objects which include pods, services, replicationcontrollers, and others.
- **Kube-controller-manager** - is a an application control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.
- **Kube-scheduler** - its job is to take pods that aren't bound to a node, and assign them one along with hardware/software/policy constraints
- **etcd** - highly-available key value store used as Kubernetes' backing store for all cluster data

Non Master Kubernetes Components

- Each individual non-master node in our cluster runs two processes:
 - **Kubelet** - which communicates with the Kubernetes Master.
 - **Kube-proxy** - A network proxy which reflects Kubernetes networking services on each node.



Kubernetes Basics



Kubernetes Basics

To work with **Kubernetes objects**—whether to create, modify, or delete them—you'll need to use the **Kubernetes API**. When you use the **kubectl** command-line interface.

<https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

Kubernetes Building Blocks

Basic Kubernetes Object

- ❑ POD
- ❑ SERVICE
- ❑ VOLUME
- ❑ NAMESPACE

Controllers

- ❑ Deployment
- ❑ StatefulSet
- ❑ DaemonSet
- ❑ JOB
- ❑ ReplicaSet

Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.

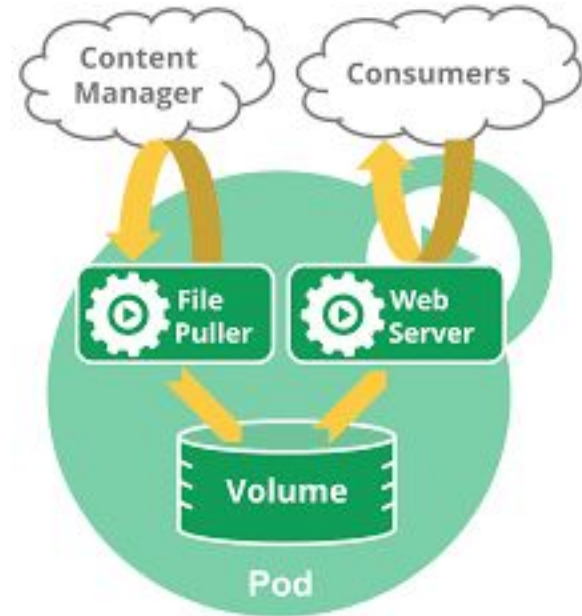
Names of resources need to be unique within a namespace, but not across namespaces.

Namespaces can not be nested inside one another and each Kubernetes resource can only be in one namespace.

Namespaces are a way to divide cluster resources between multiple users

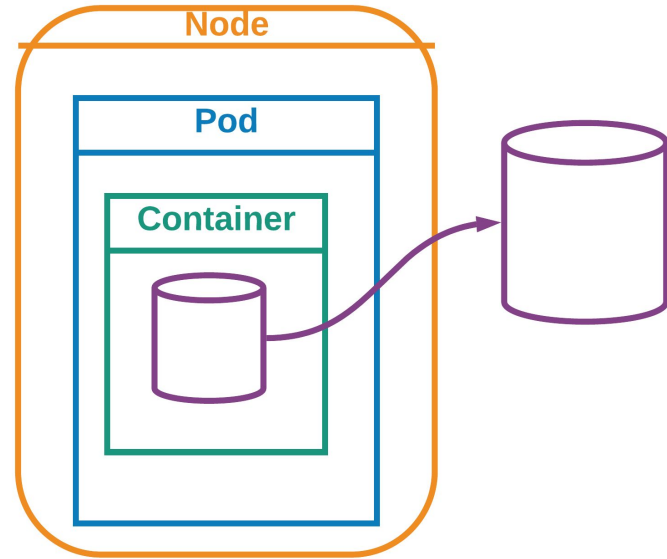
POD

- Basic building blocks.
 - Pods that run a single container
 - Pods that run multiple containers that need to work together



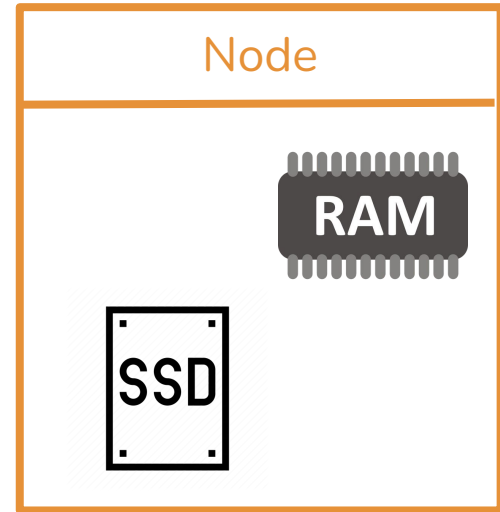
Volumes

- Persistent Volume for save state.
- When running a container together in a Pod - share files between those containers.

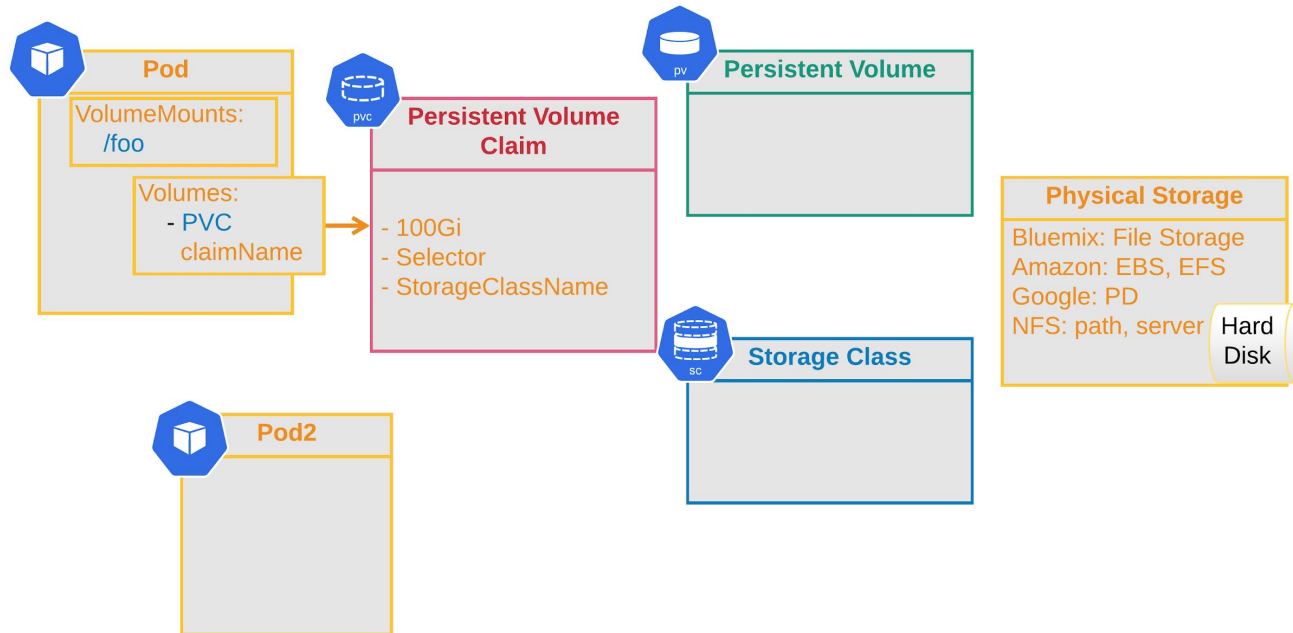


Volumes

- emptydir
- hostPath
- Cloud Volumes
- NFS
- Persistent Volume Claim
PVC



Volumes

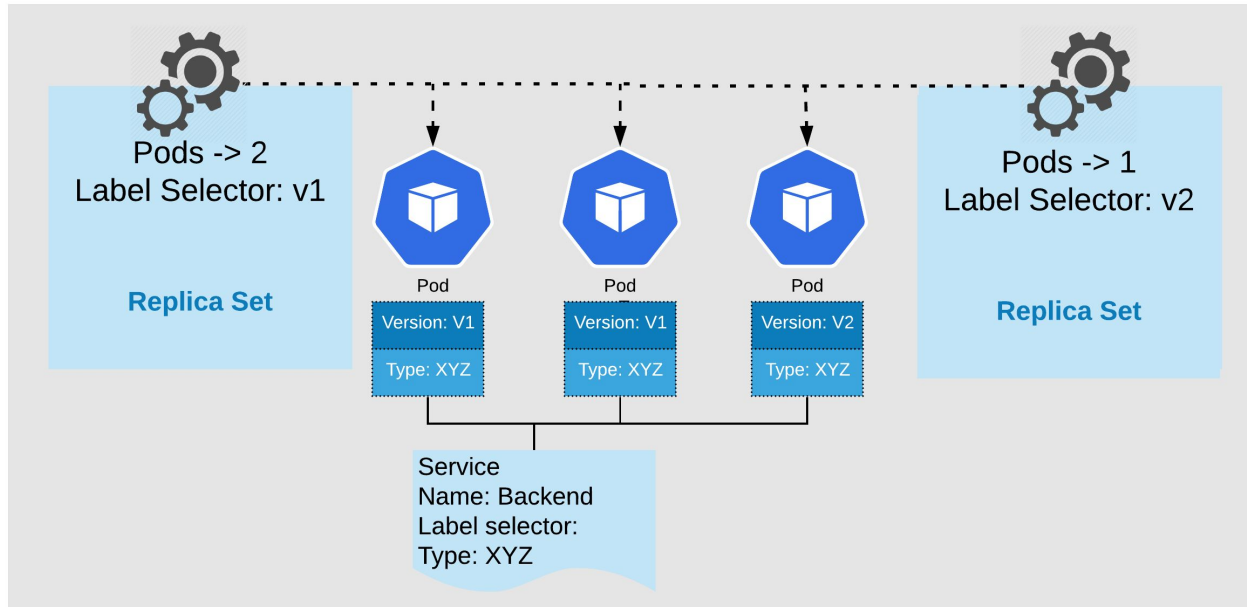


Labels and Selectors

- *Labels* are key/value pairs that are attached to objects, such as pods.
- *Labels* can be used to organize and to select subsets of objects.
- Each object can have a set of key/value labels defined. Each Key must be unique for a given object.

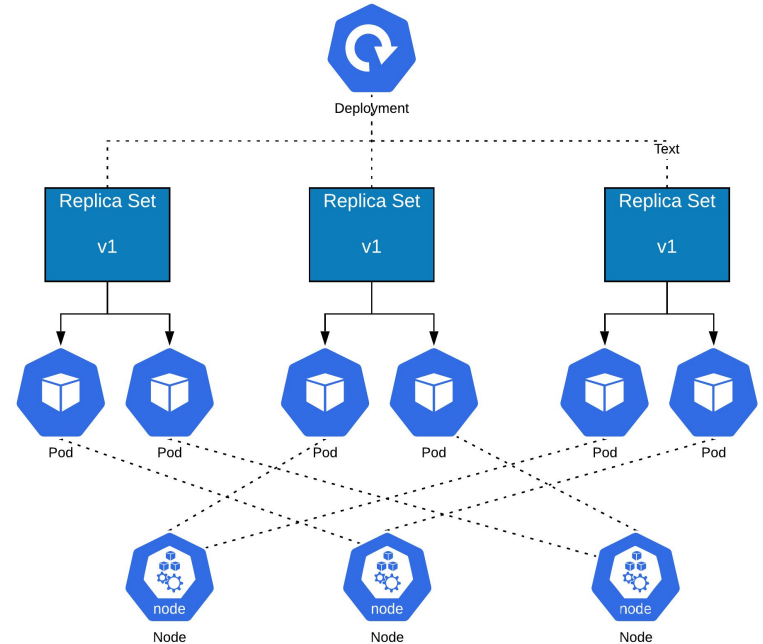
ReplicaSets

- ReplicaSet ensure how many replica of pod should be running. It can be considered as a replacement of replication controller.



Deployments

A controller that provides declarative updates for Pods and ReplicaSets. We describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate.



Deployments

The strategy object dictates the different ways in which a rollout of new software can proceed. There are two different strategies supported by deployments: Recreate and RollingUpdate.

Deployment Use - Case

- **Create a Deployment:**

A deployment is created. Once that is done the ReplicaSet automatically creates Pods in the background.

- **Update Deployment:**

A new ReplicaSet is created and the Deployment is updated. Each new ReplicaSet updates the revision of Deployment.

- **Rollback Deployment:**

Used in case when the current state of the deployment is not stable.
Only the container image gets updated.

- **Scale Deployment:**

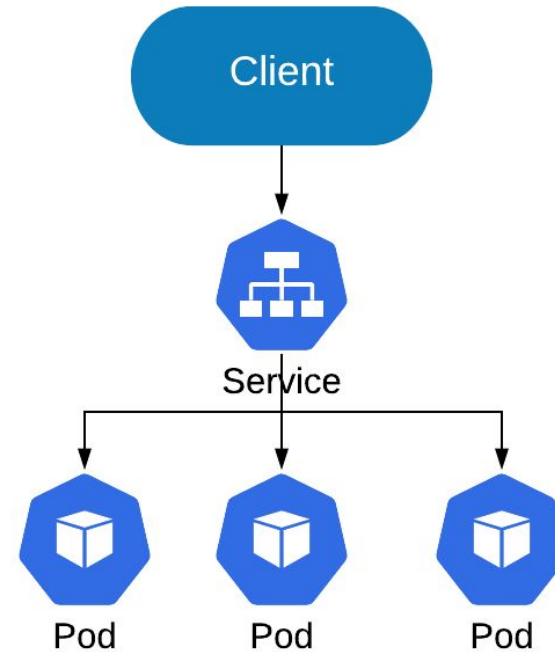
Each and every deployment can be scaled up or scaled down based on the requirement.

- **Pause the Deployment:**

Pause the deployment to apply multiple fixes and then the deployment can be resumed.

Services

- ClusterIP
- NodePort
- LoadBalancer

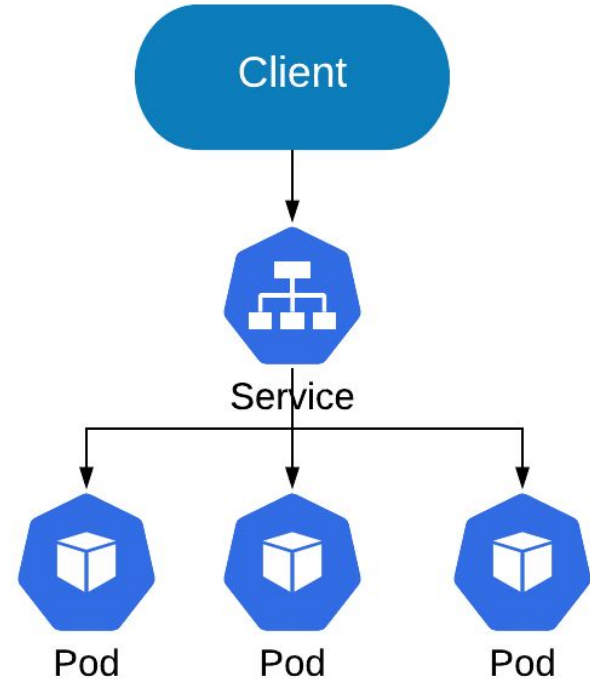


ClusterIP Service

ClusterIP

- Service **inside your cluster** that other apps inside your cluster can access.
- You can access it using k8s proxy:

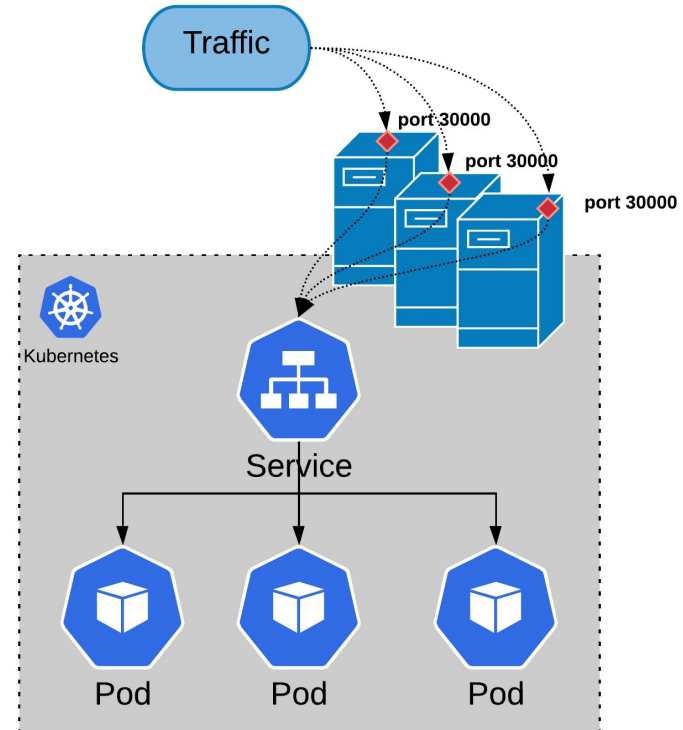
```
$ kubectl proxy --port=8080
```



NodePort Service

NodePort

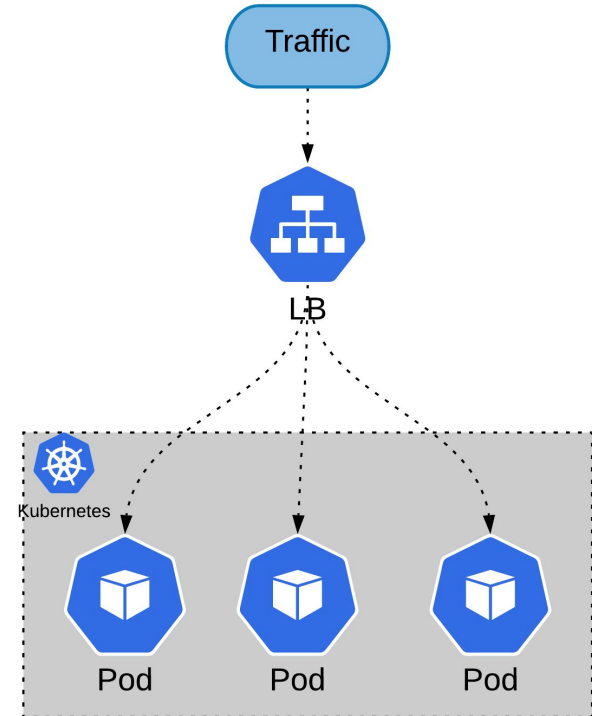
- Specific port on all the Nodes (the VMs) is open. (> 30000)
- Any traffic that is sent to this port is forwarded to the service.



LoadBalancer Service

LoadBalancer

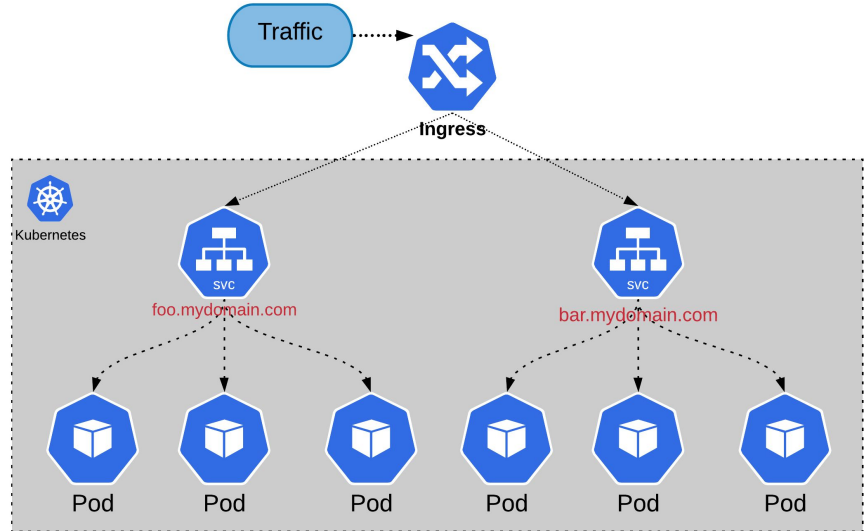
- External using network load balancer with a public IP.
- Standard way to expose a service to the internet.



Ingress

Ingress

- Ingress is a load balancer that will front multiple different pods.
- Traffic is routed via http URI or dns names.
- In GKE Ingress controller will spin up a HTTP(S) Load Balancer.





ConfigMap

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

A ConfigMap allows you to decouple environment-specific configuration from your container images, so that your applications are easily portable.

DaemonSet

A DaemonSet ensures a copy of a Pod is running across a set of nodes in a Kubernetes cluster. DaemonSets are used to deploy system daemons such as log collectors and monitoring agents, which typically must run on every node. DaemonSets share similar functionality with ReplicaSets; both create Pods that are expected to be long-running services and ensure that the desired state and the observed state of the cluster match.



Jobs - Run to Completion

A Job creates one or more Pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created.



StatefulSet

StatefulSets are valuable for applications that require one or more of the following.

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, automated rolling updates.



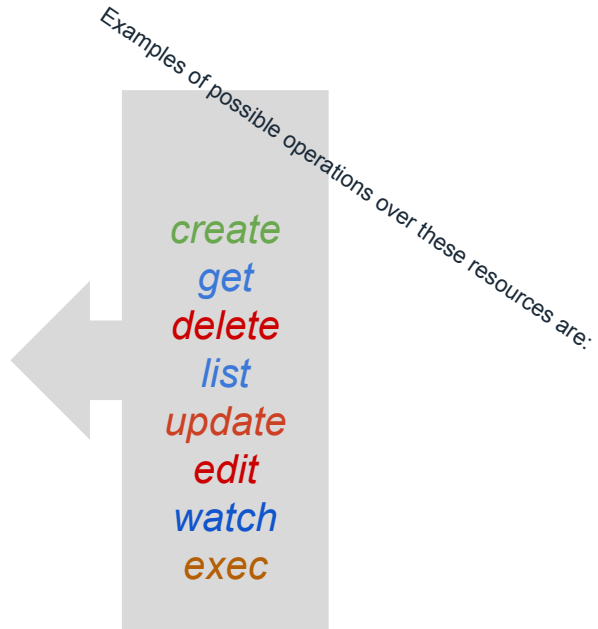
RBAC API Objects



One basic Kubernetes feature is that [all its resources are modeled API objects](#), which allow CRUD (Create, Read, Update, Delete) operations.

Examples of resources are:

Pods
PersistentVolumes
ConfigMaps
Deployments
Nodes
Secrets
Namespaces





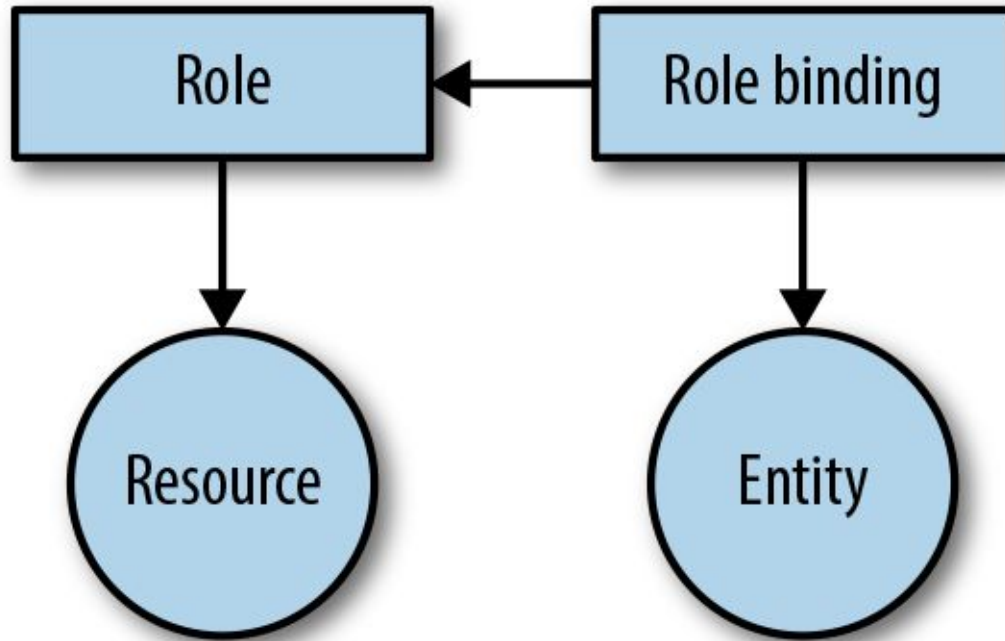
At a higher level, resources are associated with [API Groups](#) (for example, Pods belong to the *core* API group whereas Deployments belong to the *apps* API group). For more information about all available resources, operations, and API groups, check the [Official Kubernetes API Reference](#).

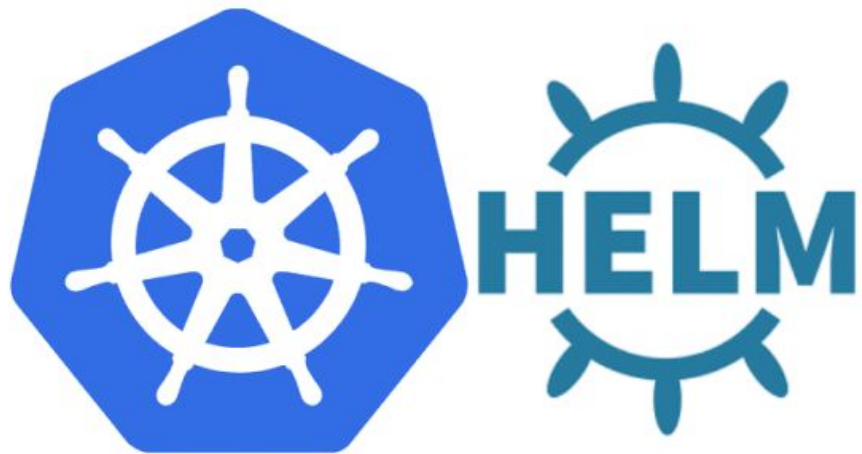
To manage RBAC in Kubernetes, apart from resources and operations, we need the following elements:

- Rules
- Roles and ClusterRoles
- Subjects
- RoleBindigs

Follow tutorial:

<https://docs.bitnami.com/kubernetes/how-to/configure-rbac-in-your-kubernetes-cluster/>





Kubernetes Helm

Helm is a tool for managing Kubernetes packages called *charts*. Helm can do the following:

- Create new charts from scratch
- Package charts into chart archive (tgz) files
- Interact with chart repositories where charts are stored
- Install and uninstall charts into an existing Kubernetes cluster
- Manage the release cycle of charts that have been installed with Helm

Three big concepts

A **Chart** is a Helm package. It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster. Think of it like the Kubernetes equivalent of a Homebrew formula, an Apt dpkg, or a Yum RPM file.

A **Repository** is the place where charts can be collected and shared. It's like Perl's [CPAN archive](#) or the [Fedora Package Database](#), but for Kubernetes packages.

A **Release** is an instance of a chart running in a Kubernetes cluster. One chart can often be installed many times into the same cluster. And each time it is installed, a new *release* is created. Consider a MySQL chart. If you want two databases running in your cluster, you can install that chart twice. Each one will have its own *release*, which will in turn have its own *release name*.